2008-03

Software reuse in the Naval Open Architecture

Greathouse, Carlus A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SOFTWARE REUSE IN THE NAVAL OPEN ARCHITECTURE

by

Carlus A. Greathouse

March 2008

Thesis Advisor:	James B. Michael
Thesis Co-Advisor:	Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2008	Master's Thesis

4. TITLE AND SUBTITLE : Software Reuse in the Naval Open Architecture	5. FUNDING NUMBERS
6. AUTHOR(S) Carlus A. Greathouse	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT (maximum 200 words)

This thesis describes a web-based continuous learning module (CLM) for use in introducing members of the Department of the Navy's acquisition community to software reuse in the context of Naval Open Architecture. The CLM introduces the student to principles for effective software reuse, explains the unique challenges of software reuse, discusses software reuse within the context of the Naval Open Architecture under the current Department of Defense and DoN policy and guidance, provides a strategy for successful software reuse, and introduces the student to the Software Hardware Asset Reuse Enterprise (SHARE) repository established by the Navy's Open Architecture (OA) program.

14. SUBJECT TERMS Software Reuse, Naval Open Architecture, SHARE,	15. NUMBER OF PAGES 95
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SOFTWARE REUSE IN THE NAVAL OPEN ARCHITECTURE

Carlus A. Greathouse
Lieutenant, United States Navy
B.S., University of Memphis, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2008

Author: Carlus A. Greathouse

Approved by: James B. Michael
Thesis Advisor

Man-Tak Shing
Thesis Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis describes a web-based continuous learning module (CLM) for use in introducing members of the Department of the Navy's acquisition community to software reuse in the context of Naval Open Architecture. The CLM introduces the student to principles for effective software reuse, explains the unique challenges of software reuse, discusses software reuse within the context of the Naval Open Architecture under the current Department of Defense and DoN policy and guidance, provides a strategy for successful software reuse, and introduces the student to the Software Hardware Asset Reuse Enterprise (SHARE) repository established by the Navy's Open Architecture (OA) program.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS

API	Application Programming Interfaces
COTS	Commercial of the Shelf
GUI	Graphical User Interface
IDS	Interface Design Specification
LCS	Littoral Combat System
NESI	Net-centric Enterprise Solutions for Interoperability
NPS	Naval Postgraduate School
NOA	Naval Open Architecture
OA	Open Architecture
PEO IWS	Program Executive Officer, Integrated Warfare Systems
PEO C4I	Program Executive Officer, Command, Control, Communications, Computers and Intelligence
PIDS	Prime Item Development Specification
ReSEARCH	Requirements Search Engine
SSDS	Ship Self Defense System
SRS	Software Requirements Specifications
SSS	System/Subsystem Specifications
SHARE	Software, Hardware Asset Reuse Enterprise
TSCEI	Total Ship Computing Environmental Infrastructure
UML	Unified Modeling Language
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis describes a web-based continuous learning module (CLM) for use in introducing members of the Department of the Navy's acquisition community to software reuse in the context of Naval Open Architecture. The CLM introduces the student to principles for effective software reuse, explains the unique challenges of software reuse, discusses software reuse within the context of the Naval Open Architecture under the current Department of Defense and DoN policy and guidance, provides a strategy for successful software reuse, and introduces the student to the Software Hardware Asset Reuse Enterprise (SHARE) repository established by the Navy's Open Architecture (OA) program.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

> The road of life twists and turns and no two directions are ever the same. Yet our lessons come from the journey, not the destination.
>
> Don Williams, Jr.

I owe a debt of gratitude to the many people who played a role in the successful completion of this thesis.

First, I have to give honor to God. Without Him life is not worth living. To my friends, coworkers, mentors, and family members who have endured the pains of this thesis work alongside me, I extend my thanks and share the pride of a completed work with them.

I would like to give special thanks to my parents and my true friends (Derrick, Charles, Calvin, Shawn and Janet). Your support has contributed greatly to my success. Thank you for your understanding listening ear and endless words of encouragement.

Finally, to my advisors, Professors Michael and Shing, SSC Charleston military staff and SSC Charleston Navigation department, thank you for working with me through some very challenging situations to get this thesis finished.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The theory and practice of software reuse continuously evolves, but one can ask: What are the factors and principles of successful software reuse? The Department of the Navy (DoN) needs to leverage the use of software reuse in its quest to deliver software-intensive systems on schedule, within budget, and with the necessary functionality.

However, before the DoN can reap the benefits of software reuse, the Navy must create, promote and positively reinforce a software engineering reuse environment. How does the DoN ensure that its investment in reuse will provide positive dividends?

The Program Executive Office for Integrated Warfare Systems (PEO IWS) identified considers the reuse of software applications as one of the key enablers for Naval Open Architecture to support of the development of National Security Systems that are *modular, interoperable, and affordable to upgrade*[1]*.* The defense community has long recognized the potential benefits that software reuse can generate, as evidenced for instance in the well-known article by Doug McIlroy that appeared in 1968[2]. Today large-scale software reuse across the industry remains elusive. Despite many setbacks, software developers now have a better understanding of the factors and principles for successful software reuse, as evident by an ever growing number of reports of successful reuse projects[3,4]. In order to increase the likelihood that the DoN will benefit from

[1] N. Guertin, Presentation to the DOD Open Technology Conference, Arlington, VA, March 14, 2007.

[2] M. D. McIlroy, "Mass produced software components," in Naur, P. and Randell, B., eds., Report on the NATO Conference on Software Engineering, NATO Scientific Affairs Division, Brussels, Belgium, 1968, pp. 138-150.

[3] M. L. Griss and A. Wosser, "Making Reuse Work at Hewlett-Packard," IEEE Software, Jan. 1995, pp. 105-107.

software reuse, Navy and Marine Corps acquisition professionals need to learn about software reuse principles and practices, in addition to learning about the linkage between software reuse and Naval Open Architecture.

This thesis describes a web-based continuous learning module (CLM) for use in introducing members of the Department of the Navy's acquisition community to software reuse in the context of Naval Open Architecture. The CLM introduces the student to principles for effective software reuse, explains the unique challenges of software reuse and discusses software reuse within the context of the Naval Open Architecture under the current Department of Defense and DoN policy and guidance. The CLM also provides a strategy for successful software reuse and introduces the student to the Software Hardware Asset Reuse Enterprise (SHARE) repository established by the Navy's Open Architecture (OA) program.

[4] A. Tomer, L. Goldin, T. Kuflik, E. Kimchi, and S. R. Schach, "Evaluating Software Reuse Alternatives: A Model and its Application to an Industrial Case Study," IEEE Transactions on Software Engineering, 30, 9 Sept. 2004, pp. 601-612.

II. BACKGROUND

A. INTRODUCTION

Software reuse promises huge returns in productivity, better quality, and a decrease in time to market for products. Software reuse is a quickly developing underlying pillar of the Naval Open Architecture Enterprise.

B. SOFTWARE REUSE

All types of software-development artifacts (e.g., use cases, requirements, designs, architectures, patterns, test cases, code, documentation, etc.) can potentially be reused if they are designed for reuse from the start. Hands-on experience has led the industry to the belief that substantial reuse can only happen in two areas: opportunistic and systematic. George Santayana's statement, "Those who cannot remember the past are condemned to repeat it", may not be completely true. The past can have triumphant moments as well. The trick is to be aware of the victorious pieces from the past to draw from for future events. This mindset applies to software engineering. Software engineering is an expanding field that has changed direction and priorities very rapidly over the last decade. Exhaustive studies and numerous projects have produced and positive results that are neither repeated nor shared across system development.

Software reuse has its roots in computer programming, with the development of software libraries containing subroutines, functions, and other reusable units of software. Today, software reuse includes the spectrum of system artifacts, including such things as requirements and software patterns.

So far in the twenty-first century, the focus has been rapid application development, fastest time to market for updates and keeping up with the Internet technology wars. The last fifty years have seen tremendous changes in software engineering. The late 1990s saw a shift from processes, tools, documentation,

negotiations and plans to individuals, collaborations, working software, and responding to change. The enterprise that can get a product or service to market first wins. Organizations have moved away from the traditional waterfall models to spiral, evolutionary, or iterative process models. Such models use the project's risk to determine how much engineering is enough. Organizations have also turned to open source development as another means of speeding up time to market and getting expansion ideas for future code capability. This century has also seen an exponential increase in the use of software by non-programmers. The increased usage has forced programmers to create much better Graphical User Interfaces (GUI). This also opened the door to more in-depth studies on Human Computer Interaction (HCI).

Software reuse is a means to achieve improvement in overall software production. A high-quality software reuse process can contribute to improved productivity, quality, and dependability, in addition to aiding the acquisition professional in better managing the schedule, cost, and performance of a program or project. An initial investment is necessary, which can be relatively large in some cases, to establish a software reuse program. However, that investment can pay for itself over time. In short, the development of a reuse program and the reuse process employed in that program can help both to reduce risk in legacy and new system developments.

According to Estublier and Vega, "Reuse is not a goal in itself; it aims at speeding up and decreasing maintenance cost."[5] They also suggest that "a really reusable component has a significant cost; therefore, to be cost effective, a reusable component must be widely REUSED."[6] In some broader definitions, reuse is the use of artifacts or components from existing systems to build new

[5] J. Estublier and G. Vega, "Reuse and variability in large scale applications," in Proceedings of the 10th European Software Engineering Conference, ACM, Lisbon, Portugal, Sept. 2005, pp. 316-325.

[6] J. Estublier and G. Vega, "Reuse and variability in large scale applications," in Proceedings of the 10th European Software Engineering Conference, ACM, Lisbon, Portugal, Sept. 2005, pp. 316-325.

ones in order to achieve the advantages of reuse. The reuse backdrop encompasses a range of possible reuse techniques (see Figure 1). Reuse is the use of existing software to build new software. Variations in the definitions range from adding words like “software knowledge” and expanding the reuse umbrella to include all aspects of a software development, such as reusing a particular software process.

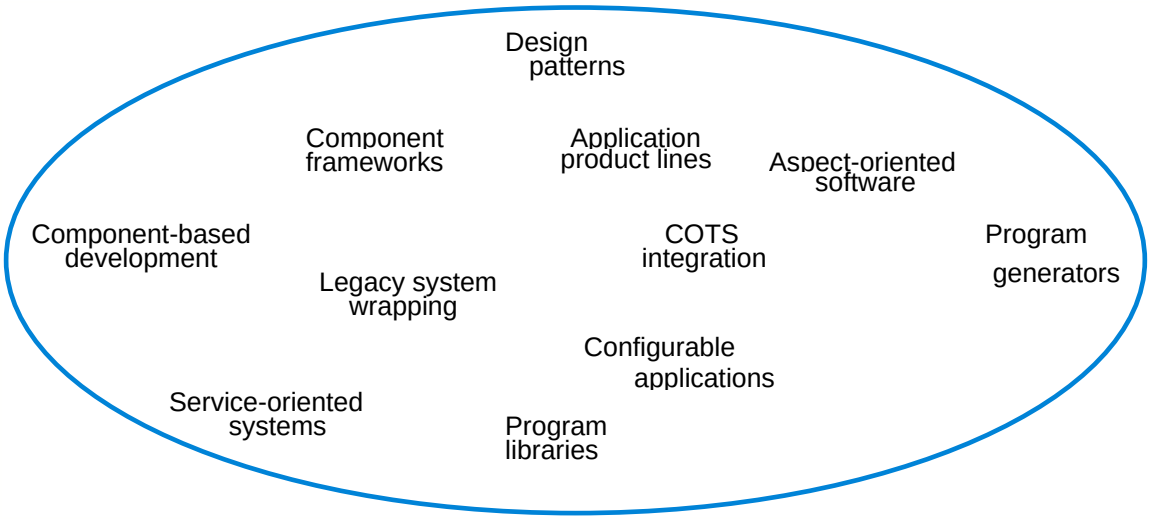


Figure 1. Reuse Backdrop. Adapted from I. Sommerville, Software Engineering, Upper Saddle River, NJ: Addison Wesley, 7th ed., 2004.

The concept behind reuse is clear-cut. Reuse emphasizes strategies, techniques and principles that enable developers to create new systems from components in repositories. Furthermore, software reuse is of interest to the Department of Defense because it is not practical for DoD to develop systems from scratch—it is too costly and time-consuming to do so. Today’s acquisition environment is one in which legacy systems and components are brought together in new developments to meet the needs of the warfigther to achieve transformation, that is, rapidly transitioning concepts and technologies to systems deployed in the battlespace. According to a spokesperson from

ComponentSource, the concept behind software reuse is a simple yet powerful one. “Reuse before you buy. Buy before you build. And if you must build anew, learn how to share.”[7]

Software reuse is divided into two types: opportunistic and systematic. Software salvage, euphemistically known as opportunistic reuse, is the unplanned reuse of system artifacts not originally designed with reuse in mind. In contrast, systematic reuse is deliberate in the sense that the artifacts are designed to be reused. The latter is the preferred type of reuse.

C. OPEN ARCHITECTURE

PEO-IWS identifies software reuse as one of the key enablers for Naval Open Architecture. As of today, large-scale industry-wide software reuse remains elusive. Within the DoN many open architectures are coming into existence. The issues involve determining the right contract language, obtaining access to reuse libraries, and the age-old issues of upgrading legacy systems coupled in software and hardware. Issues spill over into every aspect of the software development process.

Open Architecture denotes an architecture whose specifications are public. This includes officially approved standards as well as non-standard architectures whose specifications are publicly available. The opposite of open is closed or proprietary.

The great advantage of open architectures is that anyone can design add-on products for it. By making architecture public, however, a manufacturer allows others to duplicate its product. The Navy’s definition of Open Architecture (OA) is an enterprise-wide, multifaceted strategy for acquiring and maintaining National Security Systems through joint interoperable systems that adapt and exploit open-system design principles and architectures. Fundamentals of the OA

[7] M. Pepe, “Government Reduce, Reuse, Recycle Code.” CRN 23 July 2001.

strategy include increasing opportunities for competition and innovation, enabling rapidly fielded and upgradeable systems, and optimizing software asset reuse.

The Navy and Marine Corps have adopted OA as a way to reduce the rising cost and increase the capabilities of Naval warfare systems and platforms. Naval Open Architecture (NOA) allows for incorporating more COTS technology in warfare systems and enabling reuse of software and related assets. More importantly, OA will contribute to greater competition among system developers through the use of open standards and standard, published interfaces.

The definition of OA has to be stated if the Government intends to procure a system having open architecture designs and corresponding components. The contracting department must have a basis by which to understand the sharing of software and proprietary rights. It will also have to make a long list of demands on the contractor. As part of this contract process, contractors will be required to define, document, and follow an open systems approach for using modular design, standards-based interfaces, and widely supported consensus-based standards. Contractors will develop, maintain, and use an open system management plan to demonstrate compliance. The proposed open system management plan will be incorporated into the contract.

Beyond contractual language, there are standards by which code for sharing and reuse should be written. The idea behind standards is really facilitating open architecture and fast turn around of a capability. Code should be designed using industry-standard formats. Code should develop and maintain an architecture that incorporates appropriate considerations for portability, maintainability, technology insertion, vendor independence, reusability, scalability, interoperability, upgradeability, and long-term supportability. Code should be developed through an architecture that is modular and use non-proprietary or key Application Programming Interfaces (APIs). Code should be documented to describe how the proposed system architecture took into account the attempts to use non-proprietary or COTS wherever practicable. Code design

should try to minimize inter-component dependencies to allow components to be decoupled and reused, were appropriate, across various Naval programs and platforms.

Development standards create a problem. The correct way to do business is to go with the old military standards. The military standards however are very costly for the contractors to produce, causing the contractors to pass on their risk to the government in a contractual line item. The government needs contracts that can alleviate some of the burden placed on contractors while still holding them to some agreed upon standard.

The Navy should expect to expend resources to upgrade legacy systems once the processes for reuse and open architecture achieve a certain level of maturity, but the costs of the upgrades will conceivably be offset by the savings from software sharing.

Naval Open Architecture must match or exceed the rapid evolution in commercial technology. The delivery of new technologically advanced capabilities across the entire family of warfighter systems or platforms must be delivered both faster and cheaper if the military is to retain its superior warfighting capabilities in the coming years. The current process takes a decade, cost billions of dollars and only delivers in some cases minimal improvements in warfighting capability.

D. SOFTWARE, HARDWARE ASSET REUSE ENTERPRISE (SHARE)

Software, Hardware Asset Reuse Enterprise (SHARE) repository is part of the Navy's Open Architecture (OA) approach. SHARE is a resource in implementing NOA that aids in the cataloging and development of open components that include reusable software applications. SHARE provides a capability to search for, share, manage and maintain reusable assets. SHARE has a card catalog and resource library. The card catalog is web-based and

supports such actions as searching for code and sharing code. The resource library currently only contains software for Navy Surface Domain programs.

E. SUMMARY

Software reuse has become a hot topic in all aspects of the software community. A “software asset” is not simply another term for source code, but rather is anything produced during software development and designed for reuse. The DoD has seen its value and has incorporated it as part of a new strategy to acquire weapon systems. When properly applied, reuse can produce benefits, which include increased product quality and decreased product cost and schedule. The greatest benefits come from a domain specific approach, where a common set of reusable software assets act as a base for similar products production. Software reuse has substantial upfront investments, but when properly monitored can produce huge dividends.

The DoN recognizes the need for Open Architecture in system development. With an ongoing war and mounting budget constraints, there exists a need to produce software for the warfighter in an even more efficient and effective manner. The DoD is moving away from each service having its own special systems to accomplish missions. In the near future, each service will draw from a pool of similar software designs for future capabilities within that service. The Navy has delineated a specific set of OA core principles by which to operate and drive its new OA endeavors.

THIS PAGE INTENTIONALLY LEFT BLANK

III. WEB-BASED CONTINUOUS LEARNING MODULE (CLM)

A. INTRODUCTION

Software Reuse is an underlying pillar of the Naval Open Architecture Enterprise. Software reuse blends software and future capability. This approach is gaining increasing use within commercial, defense industry and government facilities as the most effective way to improve quality reduce time in getting new products to the front lines and increase productivity. It is important for Navy and Marine Corps acquisition professionals and program managers to understand these principles in order to make the correct choices and reap the benefit of software application reuse in the Naval Open Architecture. We developed this web-based continuous learning module (CLM) to teach DoN personnel about software reuse in Naval Open Architecture. The CLM introduces students to the principles for effective software reuse, explains the unique challenges of software reuse and discusses software reuse within the context of the Naval Open Architecture under the current DoD and DoN policy and guidance. The module also provides a strategy for successful software reuse and introduces students to the Software Hardware Asset Reuse Enterprise (SHARE) repository established by the Navy's Open Architecture (OA) program. The CLM contains five modules, with periodic review questions through out the course. The CLM also contains an end-of-course review and a post-training examination. The average cumulative time for course completion is two hours. Upon completion of this module, you will receive two Continuous Learning Points (CLP).

B. MODULE ONE

The first module discusses all five modules in general. In order to access all the features of the CLM, your computer must meet specific system requirements and have all the necessary software applications such as Windows Media Player and Flash Player. The computer's monitor requires the appropriate

screen settings to ensure all components from the CLM are visible. There will be help links embedded in the courseware to assist the user in correcting problems associated with viewing the course material. If the student cannot correct the problems, they will seek help from a DoD IT professional. There are consistent features that are available to you throughout the CLM (See Figure 2).

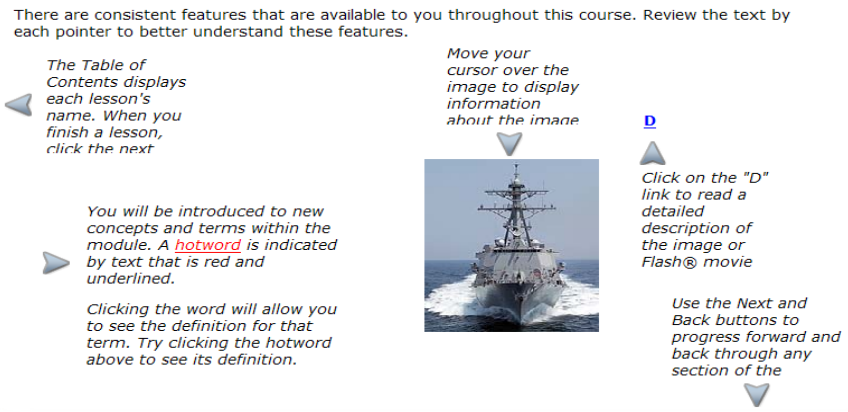


Figure 2. Navigation and Layout

The rest of the course will contain four lessons:

- Introduction to Software Reuse
- Principles of Effective Software Reuse
- Naval Open Architecture and Software Reuse
- Implementation Issues

Each lesson will have its own lesson objectives. Additionally, each module contains Knowledge Reviews throughout. These Knowledge Reviews are designed to prepare you for the end of module examination. Knowledge reviews are not graded, you may take them as many times as you would like. The exams, on the other hand, require you to demonstrate mastery of the learning objectives covered by its associated topics. The last two module features are the Summary slide and Exam. The summary slide reviews the items that you should have covered and the exam will cover questions that were presented somewhere in

the lesson. To complete this module successfully, you must pass all exams with a score of 100%. The CLMs allow for unlimited test attempts until a 100% score is obtained. There is no instructor interaction. To receive credit for this module you must complete:

- Each section of the module
- Each test
- The end of module survey

Upon completion of the CLM the student will download a certificate verifying his or her completion of the module.

C. MODULE TWO

The first lesson of the CLM is Introduction to Software Reuse. The focus of this lesson is to present an overview Software Reuse. After successful completion of this lesson, students will be able to:

- Define Software Reuse
- Describe Types of Software Reuse
- Describe Benefits and Potential Show Stoppers of Software Reuse

Software reuse is the process of creating software systems from predefined software components. Software reuse is subdivided into four types: Opportunistic, Systematic, Domain-Oriented and Strategy Driven. Opportunistic Reuse can be thought of as ad hoc or an individual effort where individual effort happens by chance. Systematic Reuse is planned or a corporate-level effort where corporate level is a repeatable process with supporting infrastructure. Domain-oriented Reuse centers around domain analysis, component development and component repository support. Strategy Driven focuses more on using software from a business prospective to cut cost, improve quality and create new business opportunities.

Software reuse has many benefits. Software reuse can reduce development cost, increase dependability of the developed system, and reduce

process risks. Software engineers do not have reinvent the wheel each time that a new project comes around; instead they can concentrate their efforts on problem-solving activities. Component interfaces are more standardized further driving down uncertainty and cost. It also adds a degree speed by avoiding customized development.

With any benefits, come potential showstoppers. Software Reuse can turn costly if extensive effort is required in adapting the reusable code. Maintaining and improving a component library will also incur cost if not done properly. Lack of tool support will cause difficultly in searching for the correct components and the integration of those components into a heterogeneous component library. Specialists tend to want to rewrite components to make them better verses using what is in the library.

D. MODULE THREE

The second lesson in the CLM discusses Naval Open Architecture and Software Reuse. The focus of this lesson is to present an overview of the DoD/DoN effort in Software Reuse. Students who successfully complete this module will be able to:

- Define Naval Open Architecture
- Summarize the Software Reuse policies from Department of Defense

Open architecture is defined as a type of computer or software architecture that is considered public. The opposite of ***closed*** or proprietary, it includes both industry standard and privately designed architectures that are made public. It allows users to see the inner workings of architecture, and allows for adding, upgrading, and swapping of components.

The Naval Open Architecture (NOA) is a multi-faceted strategy providing a framework for developing joint, interoperable systems that adapt and exploit open system design principles and architectures. It is a System Reuse framework that includes a set of principles, processes, and best practices. The

goals of NOA are to optimize total system performance, reduce difficulties in system development and upgrades, minimize total ownership costs and rapidly field affordable, interoperable systems. NOA will accomplish its goals by creating more opportunities for competition, the use of non-proprietary standards for internal interfaces, modular architectures to allow for affordable interoperability and to accommodate changing technology and requirements and finally through software reuse. Naval Open Architecture is a business process renovation, not just a changing software development practices as in Open Architecture. NOA emphasizes on the use of business drivers to produce better and less expensive software.

With the onset of the Naval Open Architecture initiative, there must be governing policies and procedures. Below is a list of the Software Reuse policies from Department of Defense:

- 12 May 2003, DoD Directive (DoDD) 5000.1, "The Defense Acquisition System"
- 5 Apr 2004, Under Secretary of Defense (Acquisition, Technology & Logistics) Memorandum, "Amplifying DoDD 5000.1 Guidance Regarding Modular Open Systems Approach (MOSA) Implementation"
- Assistant Secretary of the Navy (Research, Development & Acquisition) (ASN [RD&A]), 5 Aug 2004, OA Policy Statement, "Naval Open Architecture Scope and Responsibilities"
- Deputy Chief of Naval Operations (OPNAV) (Warfare Requirements and Program) (N6/N7), 23 Dec 2005, "Requirement for Open Architecture (OA) Implementation"

The above policies direct DoD to implement a Defense Acquisition System that will be more Flexible, Responsive, Innovative, Disciplined, and Be Streamlined and Effectively Managed.

E. MODULE FOUR

The third lesson in the CLM discusses Principles, Thinking, and Requirements of Effective Reuse. The focus of this lesson is to present

Principles for effective Software Reuse. Students who successfully complete this module will be able to:

- Describe Effective Principles
- Describe Reuse Thinking
- Describe Reuse Requirements

Effective principles in software reuse are necessary. Principles are governed through designing code with reuse in mind, which involves designing software around good design and existing components. Software components for reuse should be independent, reflect stable domain abstractions and provide access to state through interface operations. Lastly, successful and safe reuse must have a well documented design rationale and design assumptions for both the system and the software design.

Effective principles can only be as effective as the thinking associated therein. The key difference between reuse processes and conventional software engineering processes is that for reuse, the customer's requirements are modified to take advantage of what is available for reuse. While the customer may not get exactly what they want, the software should be available more economical and in a shorter time.

Once the effective principles are institutionalized, concise requirements can now be written. Requirements point to the appropriate reusable components. Reusable components should be trustworthy and behave as specified in the requirements. Components should also have associated documents to help the code specialist understand them and adapt them to new application.

F. MODULE FIVE

The fourth and final lesson in the CLM discusses Implementing Software Reuse. The focus of this lesson is to present ways to implement software reuse. Students who successfully complete this module will be able to:

- Understand the organizational and technological enablers
- Describe strategy to implement software reuse
- Understand the importance of reuse metrics
- Know how to use the SHARE repository

The first step in setting up a reuse program is to identify and develop the enablers for Software Reuse. Enablers consist of Organizational and Technological Enablers. Examples of Organizational enablers are culture, structure and polices. Examples of Technological enables are COTS, Component-based engineering and domain engineering.

The second step is to establish a strategy to gain acceptance and institutionalization of Software Reuse. The goal is to maximize the benefits of software reuse, taking into consideration resources, personnel, activities and desired outcomes. An example of a strategy may contain the following or similar steps:

- Initiate reuse program development
- Define reuse program
- Establish reuse adoption goals
- Analyze reuse adoption strategies
- Develop reuse action plan
- Implement and monitor reuse program

In summary, ensure that enablers are supported and that you are designing for reuse. Document the software components and collect metrics during the projects. There is not a “one size fits all” set of reuse metrics and metrics can be misused.

One of the United States Navy’s initiatives and a resource in implementing NOA is the Software, Hardware Asset Reuse Enterprise (SHARE) repository. It is part of the Navy’s Open Architecture (OA) approach to developing open, component that include reusable software applications as a core principle. SHARE provides a capability for discovering, accessing, sharing, managing, and

sustaining reusable assets. It has an asset library and a card catalog. Currently, it contains only one type of software specific to surface ships. Please see Appendix A for full details on the SHARE repository and its way ahead.

IV. RECOMMENDATIONS AND CONCLUSIONS

A. THE FUTURE

In 2006 several policies and procedures were promulgated directing actions be taken to move the DoN in the direction of an open architecture culture, but there is no over-whelming evidence that the policies have been embodied or taken hold in the development of Navy systems. This thesis describes a web-based continuous learning module (CLM) for use in introducing members of the Department of the Navy's acquisition community to software reuse in the context of Naval Open Architecture. It is likely that a portfolio of continuous learning modules will be developed by the DoN, incorporating the CLM described here, the Navy Reuse Enterprise.

One area of future work is to investigate how to promote the organizational acceptance of software reuse. However, in the near term, the CLM needs to be vetted by the Navy Program Executive Office for Open Architecture. In addition, the Defense Acquisition University, which will host the CLM, needs to verify module content, finalize the layout of the module, and update the DAU catalog of online courses to reflect the availability of the module.

Each of the sub modules of the CLM needs to be expanded in terms of depth of coverage. It should also capture any results from real-world successes and failures regarding NOA.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. – SHARE REPOSITORY COMPONENT SPECIFICATION: NEEDS ASSESSMENT

Jean Johnson
Naval Postgraduate School
jmjohnso@nps.edu

Introduction

The purpose of this paper is to lay the foundation for the Naval Postgraduate School SHARE component specification and ontology research project funded by Program Executive Officer, Integrated Warfare Systems (PEO-IWS). It is intended for use as a communication vehicle between the stakeholders and project performers to ensure congruence of goals and to validate requirements. First, we characterize the problem domain by describing the Software, Hardware Asset Reuse Enterprise (SHARE) repository, its contents and its unique attributes. Based on this investigation, we then provide specific recommendations for both near term and long term improvements. The near term suggestions are essentially "low hanging fruit", or ideas for quick improvements that can be implemented in a relatively short time frame. The long term improvements are associated with the benefits that can be realized once the component specification and ontology have been implemented. Finally, we outline requirements for the component specification in terms of its intended use within SHARE.

Background

In August 2006, PEO-IWS established the SHARE repository to make available combat system software and related assets to current and potential Navy contractors [5]. SHARE is one piece of the Navy's Open Architecture (OA) approach to developing modular, open systems [6], which includes reusable software applications as a core principle.

PEO IWS is currently seeking ways to improve and mature the capability provided by SHARE. Among other initiatives, two related research projects are in progress at NPS. The first, and the topic of this paper, will produce a component specification framework and ontology for use in SHARE. The component specification is essentially a model of the assets incorporated into the repository, which will enable robust search and discovery capabilities, asset submission assistance, and other repository functions. The ontology is a framework for the relationships between components, providing contextual meaning to asset descriptions. The second project will develop a prototype of a semantically-based requirements search engine (ReSEARCH) with the tools necessary to convert documents into semantically-based formal representations of requirements [4].

What is SHARE?

SHARE provides a capability for discovering, accessing, sharing, managing, and sustaining reusable assets for the Navy Surface Domain's programs [1]. SHARE consists of an asset library and a card catalog. The asset library is a collection of combat systems software and supporting artifacts. The card catalog is a web-based interface that facilitates user insight into the contents of SHARE and supports user functions including account registry, asset search and discovery, asset submission assistance, and asset retrieval requests.

The SHARE asset library is separate from the card catalog for two primary reasons. First, the majority of the contents of SHARE is classified material and therefore must be kept in a SECRET or higher container. Second, the process for retrieving assets from SHARE includes necessary steps for addressing the data rights associated with each component. For most of the components, a license agreement and Non-Disclosure Agreement are required before an asset can be issued. Due to these restrictions, the web interface and the actual assets are physically separated.

The search and discovery process in SHARE is conducted either through individual navigation of the list of assets in the catalog (see Appendix B), or by keyword search of more detailed descriptions. From the catalog list, a user can select an asset for the detailed description, which consists of identity, description and usage information if they are available. The identity information includes asset point of contact, ID, name, version, type, editor and update information. The asset description includes a free-text overview, classification level, export control and distribution statements, current state of the asset, artifact types and usage instructions. Usage information includes user agreement, subscriber, and user information.

The metadata for assets is collected during the asset submission process via an excel spreadsheet available on the SHARE user interface (see Appendix A). Submitters download the spreadsheet and then email the completed form to the SHARE helpdesk. This information includes not only contributor and asset descriptions, but also begins to address the domain specific information by identifying the asset's tie to the generic architecture provided by the Surface Navy OA Warfare Systems Architecture Element Level Decomposition.

Assets are requested from SHARE using an online interactive questionnaire. The user is asked several basic questions, such as which assets are being requested, the justification, and delivery information. The tool then prepares the necessary documents, including non-disclosure and license agreements, and provides them, along with instructions for printing and submission, to the user. Once the documents have been mailed in to the SHARE administrators, the user can track the status of the request online through the SHARE interface.

The SHARE user interface also includes some administrative information such as points of contact for the SHARE program, the list of registered users, a document library, and a calendar. There is also a place where feedback can be posted. However, this feature has not yet been utilized.

What is <u>in</u> SHARE?

The contents of SHARE are listed in Appendix B. Currently, SHARE includes the software and supporting documentation for an Aegis Baseline (7.1.1.1), the DDG1000 Total Ship Computing Environmental Infrastructure (TSCEI), and Ship Self Defense System (SSDS) Mk 2 Mod 1. For the Aegis baseline, the source code applications for all major subsystems with build files are included in the repository, as well as prime item development specifications (PIDS), computer program requirements specifications, interface design specifications (IDS), and user manuals. The TSCEI assets include both documentation and source code. SSDS submissions include the System/Subsystem Specifications (SSS), Software Requirements Specifications, (SRS) and source code for major subsystems. Additionally, the repository includes the Littoral Combat System (LCS) Open Data Model, which provides the mission architecture for LCS [2].

What makes SHARE Unique?

Several aspects of the SHARE repository make it unique when compared to any number of existing software repositories such as SourceForge [7] or Koders [3]. The first unique attribute is that the current artifacts incorporated in the database are very similar. They are each large subsystems of combat systems for Navy surface platforms. They have a similar level of granularity (very large and complex), and they are all traceable to a subset of the Surface Navy OA Warfare Systems Architecture Element Level Decomposition.

While this observation seems to point to trivial solutions for the repository, consideration of the future of the repository yields a different perspective. A primary realization is that the number of artifacts in the library will continue to grow. At some point, the number of items alone will render the search and discovery process difficult if not aided by visualization tools and robust search engines. Furthermore, if the goal of enterprise wide repository-enabled software reuse is to be realized, it is likely that the artifact characteristics will evolve over time. As Open Architecture becomes a standard development approach, more modular systems will be introduced. Once that occurs, it will be advantageous to be able to identify and retrieve modules rather than subsystems. In other words, active repository use is likely to stimulate more granular activity. Additionally, to enable enterprise-level asset sharing, the repository must support the expression of component capability and utility in a meaningful way across domains. It is also important to note that SHARE is intended include hardware artifacts, although these types of items are not currently included in the card catalog. In summary, it

is expected that over time the artifacts in SHARE will both become more heterogeneous, as well as be required to hold meaning among other more heterogeneous artifacts.

Another unique characteristic of SHARE is that there is no immediate access to assets in the repository. Due to the classification and data rights issues, we must distinctly separate the tools used for search and discovery from the components themselves. We cannot insist, for example, that the component specification become part of the component as a wrapper and expect the tools to interface with it directly. These classification and data rights issues force another important consideration. Since one of the most cumbersome processes identified for SHARE is the navigation of access authority and permissions for component retrieval, solutions aimed towards improving the usability of the repository should incorporate mechanisms for aiding in this process.

An additional distinguishing characteristic of SHARE is the part is plays in the context of the Navy enterprise. Each of the items in SHARE represent "product lines" in the surface domain, and the surface domain is a part of the larger Navy enterprise. This framework provides contextual meaning to the assets and also becomes the driving force for the desired relevance of tools developed for SHARE. Where possible, it is desirable to incorporate the domain information related to an asset to maximize its contextual meaning. Additionally, as tools are designed, developers should consider their potential use in the larger enterprise domain.

Recommendations for near term improvement

Throughout this initial research of the SHARE repository, we have identified several relatively uncomplicated improvements. These improvements can be implemented with the repository in its current state, before any fundamental framework is put in place. We offer these suggestions for consideration by SHARE leadership to enable near term enhancement of the capability. These recommendations include improved use of the metadata, increased web-site functionality, and SHARE education.

The current metadata collected for assets submitted in SHARE includes a free text overview of the asset. These descriptions are currently the best tool that users have to determine if the asset being considered is going to be valuable for them to retrieve. However, these descriptions vary greatly in the information provided. On one end of the spectrum, the descriptions provide an overview of what the component does in the system as well as information to aid in its use. On the other end, very little additional information is provided. In some cases, the acronyms that are listed in the card catalog are simply repeated. Without a better description, the user must already know a lot about the asset in order to decide if it will be useful to them.

Descriptions should be written with the assumption that the user does not already know what item(s) they are seeking. This may be a difficult perspective for program developers to take when writing summaries of their systems. Possibly, a template should be provided for the types of information required for a description in order to ensure that the appropriate level of detail is included. This description should cover what the component does, its contribution to the overall functionality of a system, and examples of how the component has been used, both in the initial system and as a reused item. Another useful item for searchers less knowledgeable about the various combat systems is an acronym list.

Several features that are popular in commercial search and discovery web interfaces such as Amazon, Google, or Netflix may also be implemented in SHARE to improve the utility of the repository. Customer reviews, frequently asked questions, and tools for visualization are integral sources of information in these web sites that could be useful in the SHARE environment as well.

The Amazon model for customer reviews could be beneficial to repository users who have identified an item that looks interesting. Amazon posts the customer ratings, a numeric assignment of quality, and also enables written feedback from the customer. For SHARE, this feedback could be tailored to answer specific questions that users would find useful. Customer feedback would include the quality assessment of the items, a description of how the customer used the component, and lessons learned regarding the item's use. As in Amazon, the SHARE tool could be set up to automatically distribute periodic emails requesting customers to review items that they have retrieved.

Information visualization aids can help people quickly identify the items of interest to them. A commonly used feature in commercial sites is the "People who bought this, also bought…" feature. This quickly points users to items they may not have been aware of, but may be relevant in solving their problem. Netflix allows you to view the details about a video in a window that pops up automatically when you move the cursor over a movie cover graphic. This feature may be helpful in navigating SHARE by allowing the user to view the detailed descriptions of components without having to click on them and wait for the information to open. Another improvement that may help provide contextual significance to repository items makes use of the reference architecture information. Currently, the link between the component and the SNOA reference architecture is collected at the time of asset submission. It may be a simple implementation to build a search interface based on this mapping. As a search option, the user could choose to display the architecture framework, and then navigate to the components in the repository by clicking on the individual module entities.

A Question and Answer (Q&A) blog could be tied to each the repository assets. Users interested in an asset would post questions that they have about

components that seem initially attractive, and asset owners post answers. Over time, the Frequently Asked Questions (FAQ) can be collected for quick reference. Also, FAQ's may reveal a lack of critical information in a component description, which can then be worked into the component metadata. The Q&A blogs themselves may provide valuable information to users as well. The same concept can also be applied to the SHARE repository overall.

Our final recommended near term improvement is less a technical solution than it is a cultural solution. One of the reasons that existing examples of reuse are successful is that people understand what they are reusing. We reuse our own code, data structures, and design patterns because we already know them and understand what they can do for us. To that end, education is critical. Before beginning a browse or search, people should understand in general what kind of information is available and how it can be used. This can be presented as a brief write-up (similar to portions of this paper) or as a simple interactive tutorial. Real examples of uses of SHARE would be valuable material to potential SHARE users as well and should be included in the information provided.

The Long Term Vision

The goal of this research is to improve the development and use of software repositories by developing a component specification designed for use in model-based applications that greatly improve the effectiveness of a software repository. We will develop a specification framework which includes a model of the components in the repository as well as the relationships that provide contextual meaning. The component models will be based on the behavior of the component as well as examples of its uses, both within the original system and in any situations where the component has been reused. The relationships may be between components within the repository, between the components and a reference or domain architecture, the component's place in the software life cycle and other relational information that will aid users in understanding the context of the component.

This framework will enable tools to be developed that will maximize the utility of the reuse repository. Two different types of tools have been identified that will be necessary to make full use of the framework. The search and discovery tools are aimed towards using the information captured in the framework to assist the user in identifying and retrieving useful items from the repository. In general, it is advantageous to provide multiple ways to search for relevant items so that users can investigate the options differently depending on their background and current needs. We envision both advanced visualization tools, such as a fish-eye graph, to aid in this process as well as tools that enable searching from available documentation (such as ReSEARCH). The second type of tool needed is tools aimed at assisting component developers by minimize the overhead of creating the component model and inserting it into the repository. One example is a specification-building tool with a wizard-type interface that will assist the

developer in creating the component specification as the component is developed. Additionally, a tool is necessary for assistance at repository-submission time to help the submitter integrate the component into the repository by building the desired relationships.

The component specification framework will incorporate all of the information that is collected through the existing efforts to collect SHARE metadata. This includes both the information collected through the current excel sheet, as well as any of the short term improvements implemented in the interim.

To support the continued and evolutionary use of the specification framework, consideration throughout development of the specification framework will be given to potentially changing aspects of SHARE as well as additional candidate repositories. As discussed previously, it is likely that the items placed in SHARE will evolve over time, from large subsystems to more granular modules. The component specification should be able to support this evolution of the contents. The framework will also be developed to support multiple repositories. While portions of the framework will contain domain specific information, the structure and non-domain specific portions should be easily portable to other repositories, along with a systematic approach to completing the domain relevant portion. Particular attention will be paid to existing DoD and other software repositories, especially those under the umbrella of the Navy OA domains such as the PEO C4I Net-centric Enterprise Solutions for Interoperability (NESI) repository. The specification framework should also support the integration of these repositories as intended by OA leadership.

Finally, it will be important to integrate the technical solutions provided by this work into the larger effort to improve software reuse within the Navy/DoD. Education, motivation and rewards are needed in order to stimulate the reuse cycle. A structured, planned and effective education campaign for these technical solutions as well as the entire domain repository effort is needed.

Requirements for component specification

Based on the initial investigation into SHARE as described in previous sections, the requirements listed here for the component specification framework are necessary to provide a solution relevant to the SHARE repository. These items will be considered throughout the framework development.

1. Improved search and discovery capability – The central focus of the specification framework is to facilitate the search and discovery process for a repository. This includes not only ease of navigation through the available components, but also completeness of the information. The goal is to educate the user about the candidate components thoroughly enough that the user knows what it is they are retrieving prior to going through that process.

2. Minimize overhead for component submission – Adding this capability to the repository will come with tradeoffs. Items must conform to the framework in order to be entered into the repository. The overhead to the asset submitters should be minimized as much as possible to avoid disuse due to unacceptable levels of difficulty. The specification framework will support tools to aid the development of the component specification for an asset and to assist integration into the repository.

3. Support multiple user perspectives – The component specification will incorporate multiple views for aiding users to reason about which components to retrieve. These perspectives include, but are not limited to:
    a. Domain specific reference architecture - Where possible, it is desirable to incorporate the available system domain information related to an asset to maximize its contextual meaning. This may be pre-existing in the form of a reference architecture or some other materials.
    b. Examples of previous uses – Examples of the components previous uses should be incorporated into the framework. This includes the component's use in the original system as well as any available examples of its reuse in later systems. Both successful and non-successful reuse examples can be included.
    c. Intra-repository component relationships – The relationships of the components in the repository to each other is also a useful view. Items that have been used in the same system or used to perform similar functions in different systems can be grouped together. Additionally, the threads of components that have been reused and reinserted back into the repository as part of a new system should be traceable.
    d. Life cycle activity information – Information about the life cycle phase or activity that the artifact is intended to support is useful as well. For example, a user may wish to search for all requirements documentation for systems that perform similar functions to their intended new system as a useful reference.

4. Support for security requirements – Due to the classified nature of the assets in SHARE, the interface for search and retrieval must be kept separate from the assets themselves. Therefore, the specification model must support this constraint of separate locations. Additionally, the metadata of classified elements must be constrained to unclassified material, and possibly include pointers to classified descriptions.

5. Support for legal concerns – As discussed in the previous sections, one of the primary difficulties specific to SHARE is the navigation of access authority and permissions for component retrieval. Any solution provided

must take into account these constraints and should incorporate mechanisms for aiding in this process wherever possible.

6. Extensible to other domains – Since SHARE is part of a greater effort to improve software reuse across the DoD, the component specification framework should support this goal. To that end, the framework should be extensible to the other domains under the Navy OA construct and will support the integration of these capabilities. Additionally, as supporting tools are designed, developers should consider their potential use in the larger enterprise.

7. Scalable for repository evolution – The specification framework should support the evolution of the repository, both from the perspective of the expected growth in the number of components contained as well as the progression towards less homogenous contents (smaller modules vs. large subsystems, various asset types – design artifacts, documentation, etc.). Additionally, the models should be capable of representing hardware artifacts that may be included as assets in the repository in the future.

8. Use of de facto standards – Wherever possible, implementation of the component specification framework will employ de facto standards such as the Unified Modeling Language (UML), Extensible Markup Language (XML), or others in order to promote broader applicability of existing tools as well as open an unbiased competition for tools to be developed.

Future Work

This paper is the first in a series of intermediate products related to the development of the component specification and ontology. Future writings will cover the results from an ongoing survey of SHARE users and other feedback that has been collected, case studies outlining success and failure stories, and intermediate deliverables supporting the larger task. Near term research activities will be focused on existing research and practical applications of repository submission procedures, repository management tools, component specification, and model driven software development (particularly, what models are used during various phases of software development) to determine if there are existing solutions that will be relevant in accomplishing the goals of the project.

References

[1] Belcher, M., “PEO IWS Software Hardware Asset Reuse Enterprise (SHARE)”, Information brief, 2007.

[2] Fein, Geoff, “Navy’s SHARE Repository Seeing Steady Growth in First Six Months”, Defense Daily, Feb 2007.

[3] Koders, www.koders.com, accessed 7 Oct 07.

[4] Martel, C., “ReSEARCH: A Requirements Search Engine”, proposal for Future Combat Systems Open Architecture, 2007.

[5] PEO-IWS Library, Software, Hardware Asset Reuse Enterprise (SHARE) Special Notice DON-SNOTE-070206-005, 6 Feb 07

[6] SHARE Home Page, https://viewnet.nswc.navy.mil/PNB/share/share.nsf/homepage?openform, accessed 7 Oct 07.

[7] SourceForge, www.sourceforge.net, accessed 8 Oct 07.

Acronyms

IDS	Interface Design Specification
LCS	Littoral Combat System
NESI	Net-centric Enterprise Solutions for Interoperability
NPS	Naval Postgraduate School
PEO IWS	Program Executive Officer, Integrated Warfare Systems
PEO C4I	Program Executive Officer, Command, Control, Communications, Computers and Intelligence
PIDS	Prime Item Development Specification
ReSEARCH	Requirements Search Engine
SSDS	Ship Self Defense System
SRS	Software Requirements Specifications
SSS	System/Subsystem Specifications
SHARE	Software, Hardware Asset Reuse Enterprise
TSCEI	Total Ship Computing Environmental Infrastructure
UML	Unified Modeling Language
XML	Extensible Markup Language

Appendix – SHARE Asset Contribution Form

UNCLASSIFIED			
SHARE Asset Contribution v6			
Complete the yellow areas below, and return to: HelpDesk@Nice-Help.net, with cc: to melody.belcher@navy.mi, and gregory.hartwig@navy.mil (Items with gray-fill labels will not be published)		SHARE (assigned by Help Desk) Control Number:	
	Asset Name:		
	Asset Description:		
	Request Date:		
Contributor:	Name:		
	Phone:		
	E-Mail ID:		
	Organization:		
	Mailing Address:		
Government Major Program Manager (MPM):	Program Title:		
	Name:		
	Phone:		
	E-Mail ID:		
	Organization + Code:		
	Approval:		
MPM Alternate:	Name:		
	Phone:		
	E-Mail ID:		
	Organization + Code:		
Rationale for Contribution:			
Impacts:			
Asset:	Asset Type:	Sub-Type:	Populate one selection below with a description of the type of asset
	Tactical Application	System	
		Application Program	
		Package	
		System Service	

		Component(s)	
		Library	
		Module/Code Fragment	
		Database / Data Files	
	Development Support	Framework	
		Tools / Utilities	
		Test Tools/ Environments	
	Non-code	Enterprise Framework	
		Data Architecture	
		Pattern / Design / Algorithm	
		Standard / Interface / API	
	New, Modified, or Linked:		
	Dependencies on other assets, COTS, etc.		
	Version:		
	Description:		
	Date of Asset:		
	Target OS:		
	Acquisition or Final?		
	Test Level:		
	Certification Level:		
	OACE Level (self assessment):		
	OAAT Level (self assessment):		
	Complete?		
	Buildable?		
	Planned Updates:		
	Usage Instructions:		
Types of artifacts included within the asset:			
		Included? (Y/N)	Format (e.g., DOORS, MS-Word, etc.)
Requirements:	Requirements Specification:		
	Requirements Database:		
Design:	Design Models		
	Design Documents:		

	Patterns:	
	Algorithms:	
	White Papers:	
	Data Models:	
	Simulation Models:	
Code:	Source Code:	
	Compiled Libraries:	
	Executable Programs:	
Test:	Test Plan:	
	Test Procedures:	
	Test Results:	
	Test Tools/Scripts:	
	Test Source Data Files:	
	Test Truth Data:	
	Simulators:	
Interface:	IRSs/IDDs	
	IDSs	
	APIs	
Architecture:	Architecture Model:	
	Architecture Document:	
Supporting Artifacts:	User Documentation:	
	Training Documentation:	
	Build Scripts/Instructions:	
	Other:	
Software Programming language and Operating System(s) Supported		
	Pgm Language(s):	
	Run time Environment(s):	
Media Description:	Security Classification:	
	Program's Security Classification Guide ID#:	
	Has MPM pre-approved classification release authority?	
	Media Format:	
	Number of Files:	
	Structure of Files:	
	Total Data Size:	

	Element	Applic.? (Y or blank)	Notes
Architectural Elements (check all that apply):	**Middleware/OS**		
	Host Application		
	Infrastructure Services		
 Microsoft PowerPoint Slide	**Intelligence**		
	Track Management		
	Common C2 Services		
	Operational C2		
	Tactical C2		
	Mission Planning		
	Resource Management		
	NTM Tasking/Status		
	Common Display Services		
	Common Operator Displays (e.g., GUIs)		
	Plaform Specific Operator Displays		
	Plaform Specific Display Devices		
	Local & Offboard Sensor Control		
	Sensor Adaptation		
	Sensor		
	Sensor Stimulation / Simulation		
	Communications Control		
	Communications Adaptation		
	Communications Devices		
	EXCOMM Simulation / Stimulation		
	Off-board Organic Vehicle Control		
	Off-board Organic Vehicle Adapation		
	Off-board Organic Vehicle		

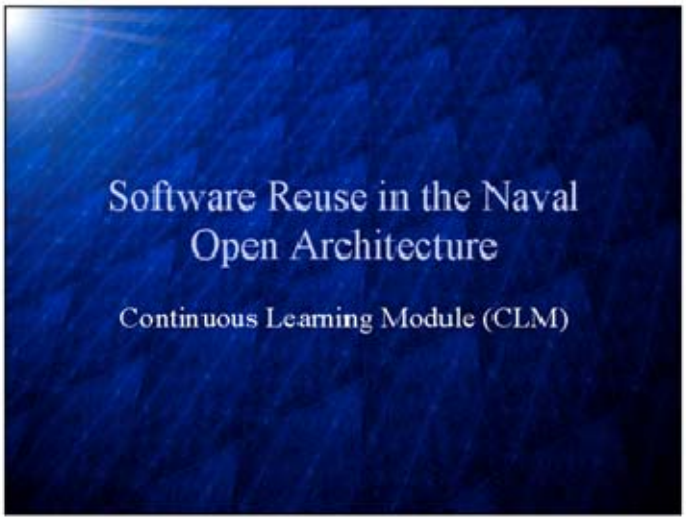
	Vehicle Simulation / Stimulation	
	Weapon Control	
	Weapon Adaptation	
	Weapon	
	Weapon Simulation / Stimulation	
	Specialized Trainer	
	Ship Control	
	Computing Hardware	
	Engineering / Damage Control	
	Readiness / Support Adaptation	
	Training Control	
	Training Assessment	
	Training Dev. Env.	
	Readiness / Support	
Distribution Statement:		
Data Rights Markings:		
Commercial Software:		
Special Licenses:		
Open Source Software Licenses:		
Data Rights Assertions:		
Any Additional Information:		

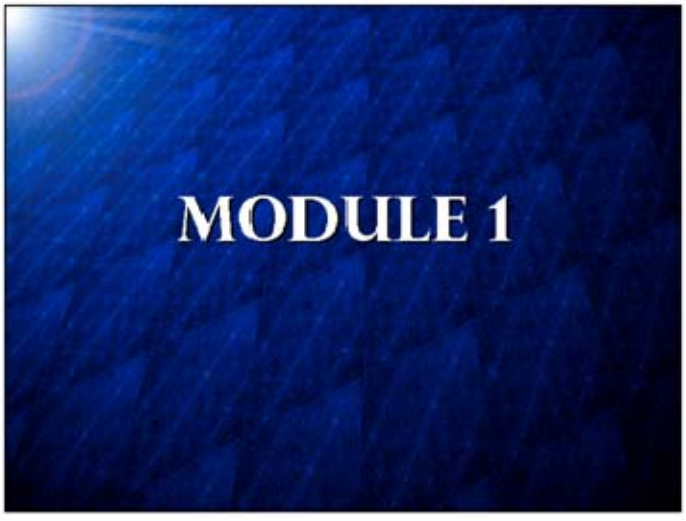
Appendix – SHARE Contents (as of 07 Oct 07)

Name	State	Type	POC	Version
AEGIS				
A-spec: WS-21200/5 SCN 1	Available	Documentation	Andy Li	7.1.1.1
B1-specs: ACTS WS-33417/2	Available	Documentation	Andy Li	7.1.1.1
B1-specs: ADS WS-10666/4	Available	Documentation	Andy Li	7.1.1.1
B1-specs: C&D WS-21208/6	Available	Documentation	Andy Li	7.1.1.1
B1-specs: FCS WS-10521/7	Available	Documentation	Andy Li	7.1.1.1
B1-specs: ORTS WS-10523/10	Available	Documentation	Andy Li	7.1.1.1
B1-specs: SPY WS-10520/10	Available	Documentation	Andy Li	7.1.1.1
B1-specs: WCS WS-10522/9	Available	Documentation	Andy Li	7.1.1.1
B5-specs: TCP WS-33419/2A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
B5-specs: ADS WS-21366/4A VOL 1-41	Available	Documentation	Andy Li	7.1.1.1
B5-specs: C&D WS-21240/4A VOL 1-28	Available	Documentation	Andy Li	7.1.1.1
B5-specs: FCS WS-10557/12A	Available	Documentation	Andy Li	7.1.1.1
B5-specs: ORTS WS-21234/6A	Available	Documentation	Andy Li	7.1.1.1
B5-specs: SPY WS-10554/16A VOL 1-3	Available	Documentation	Andy Li	7.1.1.1
B5-specs: WCS WS-10555/17A VOL 1-6	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: NAV/AWS S9427-AN-IDS-020/WSN-7	Available	Documentation	Andy Li	7.1.1.1 (31 July 1997)
IDS-specs: WCS/SPY WS-19632/10A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: SPY/SPY SIG PRO WS-19634/8A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: FCS/FCS DCC WS-19640/4A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ORTS/WCS WS-19644/10A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ORTS/SPY 19646/12A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: WCS/LAMPS WS-19657/1	Available	Documentation	Andy Li	7.1.1.1. (01 Mar 2000)
IDS-specs: ACTS/SPY WS-19681/8A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ACTS/WCS WS-19682/10A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ADS/ORTS WS-21267/2A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ADS/C&D WS-21272/2A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ORTS/ACTS WS-21278/2A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ADS/ACTS WS-21286/2A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ORTS/SCA WS-21287/1A	Available	Documentation	Andy Li	7.1.1.1

IDS-specs: ACEG/AP WS-21288A PT 1-5	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: AP/AOCD WS-21290/1A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: SPY/C&D WS-21327/8A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: C&D/WCS WS-21328/7A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ORTS/C&D WS-21329/6A	Available	Documentation	Andy Li	7.1.1.1
IDS-specs: ACTS/C&D 21338/7A VOL 1-2	Available	Documentation	Andy Li	7.1.1.1
S/W: Aegis C&D source code	Available	Application	Andy Li	7.1.1.1
S/W Aegis FCS source code	Available	Application	Andy Li	7.1.1.1
S/W Aegis WCS source code	Available	Application	Andy Li	7.1.1.1
S/W Aegis SPY source code	Available	Application	Andy Li	7.1.1.1
Aegis Quick Reference Guides (QRGs)	Available	Documentation	Andy Li	7.1.1.1
Aegis Interface Design Specifications (IDSs)	Available	Documentation	Andy Li	7.1.1.1
Aegis Interface Design Specs / ACD-9072_3	Available	Documentation	Andy Li	7.1.1.1
Aegis Interface Design Specs / WS-10512-2A	Available	Documentation	Andy Li	7.1.1.1
Aegis Reusable Components (ARC) User Manuals	Available	Documentation	Andy Li	7.1.1.1
S/W: Aegis C&D build/support files	Available	Code	Andy Li	7.1.1.1
S/W: Aegis Reusable Components (ARC)	Available	System Service	Andy Li	7.1.1.1
DDG 1000				
TSCEI 4.1 Documentation	Acquisition	Documentation	Tom Kostyo	4.1
TSCEI 4.1 Source Code	Acquisition	Application	Tom Kostyo	4.1
TSCEI 4.2.2 Documentation	Acquisition	Documentation	Tom Kostyo	4.2
LCS				
LCS Data Model 2006-11-22	Acquisition	Architecture/Design	Belcher_MelodyS	11/22/2006
LCS Open Data Model Package - 5/22/2007	Acquisition	Architecture/Design	NA	3/20/2007
SSDS				
SSS: SSDS MK 2 System/Subsystem Specification	Available	Documentation	Andy Li	MK 2 Mod 1
SRS: Display Services	Available	Documentation	Andy Li	MK 2 Mod 1
SRS: Human Machine Interface	Available	Documentation	Andy Li	MK 2 Mod 1
SRS: Infrastructure Services (IS)	Available	Documentation	Andy Li	MK 2 Mod 1
SRS: Tactical Operations (TO)	Available	Documentation	Andy Li	MK 2 Mod 1
S/W: Tactical Operations Function	Available	Application	Andy Li	MK 2 Mod 1
S/W: OL	Available	Application	Andy Li	MK 2 MOD 1

APPENDIX B– SOFTWARE REUSE IN THE NAVAL OPEN ARCHITECTURE





1

Module Systems Requirements

In order to access all features of this module, your computer must meet specific system requirements and have the necessary software applications.

Monitor Screen Resolution

Flash Player

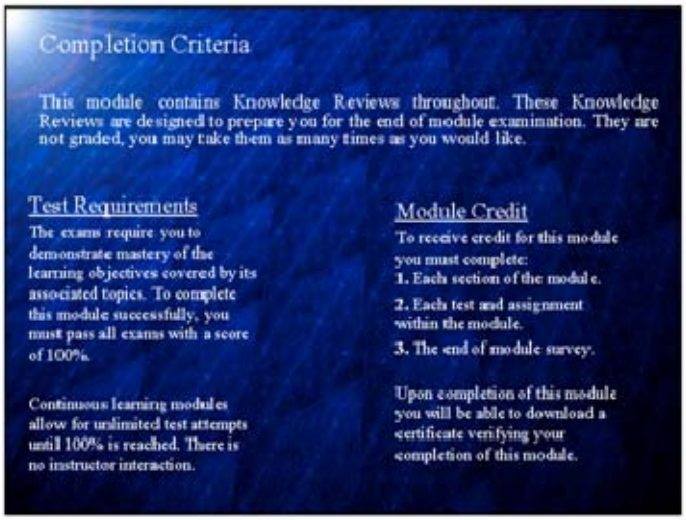
Window Media Player

Accessibility

Additional Support

2


Module Objectives
This module consists of four lessons. Click each topic for more information on that section of the module.
• Introduction to Software Reuse
• Principles of Effective Software Reuse
• Naval Open Architecture and Software Reuse
• Implementation Issues

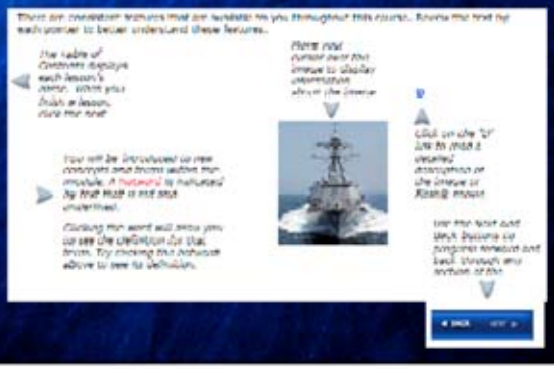

Completion Criteria
This module contains Knowledge Reviews throughout. These Knowledge Reviews are designed to prepare you for the end of module examination. They are not graded, you may take them as many times as you would like.
Test Requirements
The exams require you to demonstrate mastery of the learning objectives covered by its associated topics. To complete this module successfully, you must pass all exams with a score of 100%.
Continuous learning modules allow for unlimited test attempts until 100% is reached. There is no instructor interaction.
Module Credit
To receive credit for this module you must complete:
1. Each section of the module.
2. Each test and assignment within the module.
3. The end of module survey.
Upon completion of this module you will be able to download a certificate verifying your completion of this module.

Knowledge Review

- Each lesson of the module will contain questions.
- **Note:** These questions are not assigned any point value and should not be confused with Exam questions which must be completed to obtain credit for the course.
- The Knowledge Review demonstrates a typical multiple choice question.

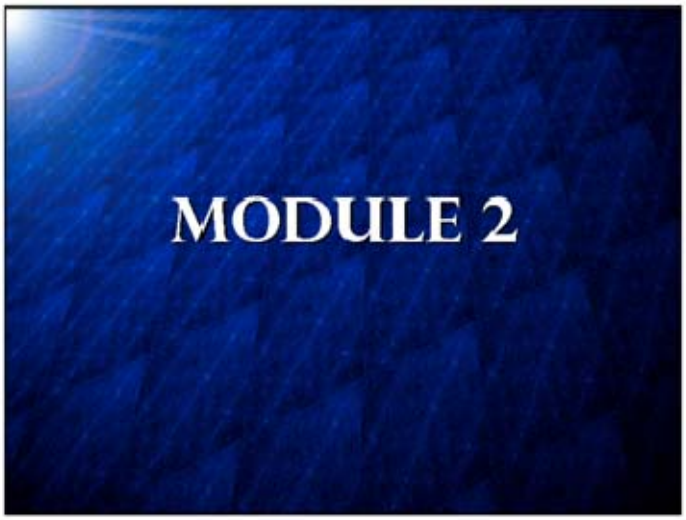
Navigation and Layout

There are consistent features that are available to you throughout this course. Review the text by each pointer to better understand these features.

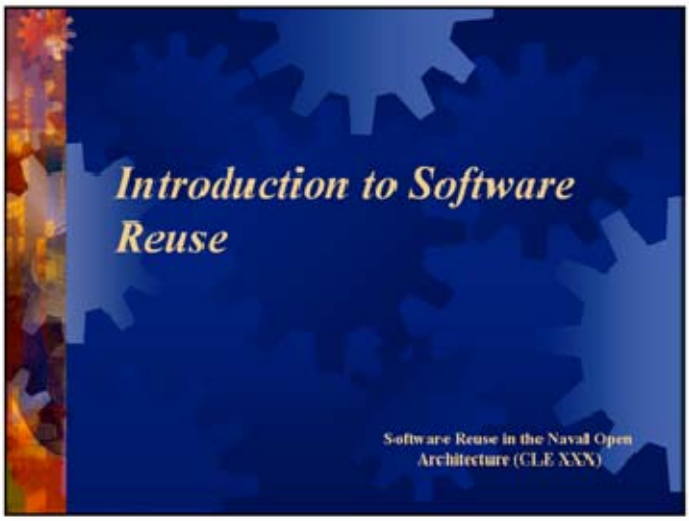


4


Summary
In this module, you learned:
• System Requirements
• Module Objectives
• Completion Criteria
• Navigation and Layout.


MODULE 2

5



Lesson Objectives

- The focus of this module is to present an overview of Software Reuse.
- Students who successfully complete this module will be able to:
  - Define Software Reuse
  - Describe Types of Software Reuse
  - Describe Benefits and Potential Show Stoppers in Software Reuse

What is Software Reuse?

- Software reuse is the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve, and adapt software artifacts for the purpose of using them in its development activities. (Mili et.al. 2002)
- "The use of an asset in the solution of different problems."
  - IEEE Std 1517-1999. *IEEE Standard for Information Technology—Software Life Cycle Processes—Reuse Processes*. Institute of Electrical and Electronics Engineers, Inc., 1999, p. 3.

Types of Software Reuse

- Opportunistic Reuse (Salvage)
  - ad hoc, individual effort
- Systematic Reuse
  - planned , corporate-level effort

Opportunistic Software Reuse (Software Salvage)

- Individual effort, happens by chance, usually without tool support

Systematic Software Reuse

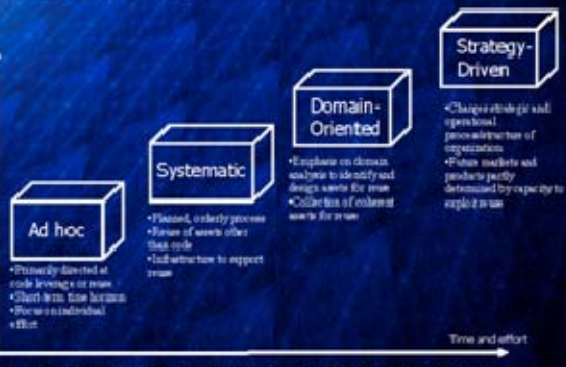
- Corporate level planned effort, a repeatable process with infrastructure support

8

Systematic vs. Opportunistic reuse

- Systematic
  - Organized
    - The practice of reuse according to a well-defined, repeatable process
  - Planned
    - The practice of reuse according to a long-term plan
- Opportunistic
  - Ad Hoc
    - The practice of reuse without well-defined process
  - Unplanned
    - The practice of reuse in an informal way without any organization-wide reuse strategy

Levels of reuse

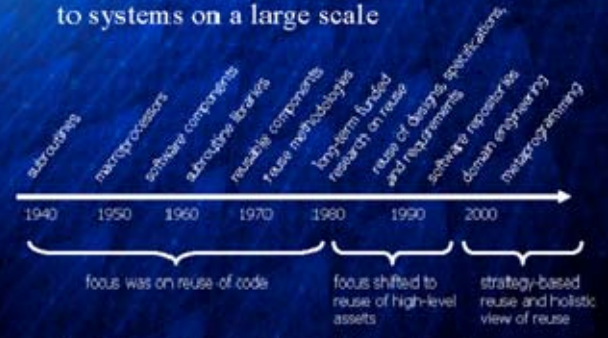


From W. C. Lim, *Managing Software Reuse*, Upper Saddle River, N.J.: Prentice Hall, 1998, p. 166.

9

Software Reuse Timeline

- Reuse moved from programs on a small scale to systems on a large scale



N.B.: The U.S. DoD has played a leading role in promoting software reuse.

Domain-oriented Reuse

- Domain-oriented reuse involves the following activities:
  - Domain analysis
    - define application domain to be investigated
    - categorize items extracted from domain
    - collect representative applications from the domain
    - analyze each application from sample
    - develop an analysis model for objects
  - Reusable component development
    - design and develop software reusable components
- Reusable component repository support
  - create and maintain reusable component repository

Strategy Driven Reuse

- Migrate reuse into the company's business in order to:
  - Cut costs and improve the quality of the software
  - Increase agility
  - Create new business opportunities

Benefit of Software Reuse (1/2)

- Decrease in Cost
- Increased Reliability
  - components already exercised in working systems
- Reduced Process Risk
  - less uncertainty in development costs

Benefit of Software Reuse (2/2)

- Effective Use of Specialists
  - *"avoid reinventing the wheel"*
- Standards Compliance
  - Component interfaces will be more standardized
- Accelerated Development
  - avoid custom development and speed up delivery

Potential Show Stoppers (1/2)

- Increase development cost
  - Will happen if extensive effort is required in adapting the reusable code
- Lack of tool support
  - Will be very difficult to search for the correct components and to integrate the components from a heterogeneous component library system
- The Not-Invented-Here Syndrome
  - Some software engineers prefer to rewrite components as they believe that they can do a better job

Potential Show Stoppers (2/2)

- Continue investment in maintaining a component library
  - Improper understanding of the ongoing needs to maintain and evolve the reusable component libraries
- Difficulties in finding the correct reusable components
  - It may be very difficult to select the correct component for reuse because the proper functioning of a reusable component often depends on the new environment which the component will be working in

Summary

In this module, you learned to:

- Define Software Reuse
- Describe Types of Software Reuse
- Describe Benefits and Potential Show Stoppers of Software Reuse

End of Lesson Questions

- What can be reused?
- Name one benefit to reuse?
- Name two potential show stoppers of reuse?
- What is this the definition of : "use of existing software artifacts in the development of other software" ?

MODULE 3

Lesson Objectives

- The focus of this module is to present an overview of the DoD/DoN effort in Software Reuse.
- Students who successfully complete this module will be able to:
  - Define Naval Open Architecture
  - Summarize the Software Reuse policies from Department of Defense

15

Open Architecture (**OA**)

- Open architecture is defined as:
  - A type of computer or software architecture that is considered public.
    - The opposite of *closed* or proprietary
  - Includes both industry standard and privately designed architectures that are made public.
  - Allows users to see the inner workings of architecture, and allows for adding, upgrading, and swapping of components.

Naval Open Architecture (NOA) (1/4)

- The Naval Open Architecture is defined as:
  - A multi-faceted strategy providing a framework for developing joint, interoperable systems that adapt and exploit open system design principles and architectures.
  - It is a System Reuse framework that includes a set of principles, processes, and best practices.

16

Naval Open Architecture (NOA) (2/4)

- The goals of NOA include:
  - Optimize total system performance
  - Reduce difficulties in system development and upgrades
  - Minimize total ownership costs
  - Rapidly field affordable, interoperable systems

Naval Open Architecture (NOA) (3/4)

- NOA will accomplish its goals through
  - More opportunities for competition
  - The use of non-proprietary standards for internal interfaces
  - Modular architectures to allow for affordable interoperability and to accommodate changing technology and requirements
  - Software reuse

Naval Open Architecture (NOA) (4/4)

- Naval Open Architecture is a business process renovation, not just a changing software development practices as in Open Architecture
  - NOA emphasizes on the use of business drivers to produce better and less expensive software

DoD Policies on Open Architecture

- 12 MAY 2003, DoD Directive (DoDD) 5000.1, "The Defense Acquisition System"
- 5 APR 2004, Under Secretary of Defense (Acquisition, Technology & Logistics) Memorandum, "Amplifying DoDD 5000.1 Guidance Regarding Modular Open Systems Approach (MOSA) Implementation"

The above policies direct DoD to implement a Defense Acquisition System that will be more:
Flexible, Responsive, Innovative, Disciplined, and Be Streamlined and Effectively Managed

DoN Policies on Naval Open Architecture (1/4)

- Assistant Secretary of the Navy (Research, Development & Acquisition) (ASN [RD&A])
  - 5 AUG 2004, OA Policy Statement, "Naval Open Architecture Scope and Responsibilities"
- Deputy Chief of Naval Operations (OPNAV) (Warfare Requirements and Program) (N6/N7)
  - 23 DEC 2005, "Requirement for Open Architecture (OA) Implementation"

DoN Policies on Naval Open Architecture (2/4)

- ASN's memo set out OA policy, established an OA Enterprise Team (OAET), and assigned its roles and responsibilities:
  - Lead the Navy Enterprise to OA implementation
  - Provide OA Systems Engineering leadership to Program Executive Office's (PEOs), industry partners, Joint Organizations, Navy Warfare Centers and other participating organizations

DoN Policies on Naval Open Architecture (3/4)

- Provide the forum and process by which cross domain OA proposals and solutions are reviewed and approved
- Oversee OA implementation efforts ensuring standardized and disciplined processes are utilized across domains
- Identify cross-domain components and opportunities for cost reduction and reuse
- Leverage technical, business, and organizational solutions from all participating communities

DoN Policies on Naval Open Architecture (4/4)

- DCNO's memo establishes OPNAV requirement to implement Open Architecture principles across the Navy Enterprise
- Bottom line is that there is a organizational shift to embrace open architecture and software reuse in Naval system acquisition

20

Summary

In this module, you learned:

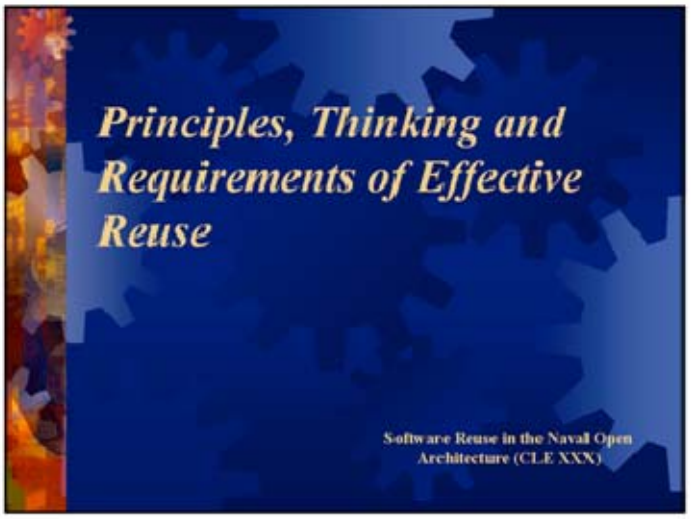
- What Naval Open Architecture is
- The DoD/DoN policies on Naval Open Architecture

End of Lesson Questions

- What is the differences between the Open Architecture and the Naval Open Architecture?
- What role will Software Reuse play in the Naval Open Architecture?

21

MODULE 4


Principles, Thinking and Requirements of Effective Reuse
Software Reuse in the Naval Open Architecture (CLE XXX)

22

Lesson Objectives

- The focus of this module is to present Principles for effective Software Reuse.
- Students who successfully complete this module will be able to:
  - Describe Effective Principles
  - Describe Reuse Thinking
  - Describe Reuse Requirements

Effective Principles

- Design with reuse involves designing software around good design and existing components
- Effective Principles
  - Software components for reuse should be independent, reflect stable domain abstractions and provide access to state through interface operations
    - Application families are related applications developed around a common core for software reuse
  - Successful and safe reuse must have a well documented design rationale and design assumptions for both the system and the software design.

Reuse Way of Thinking

- The key difference between reuse processes and conventional software engineering processes is that for reuse, the customer's requirements are modified to take advantage of what is available for reuse.
- While the customer may not get exactly what they want, the software should be available more economical and in a shorter time

Reuse Requirements

- Must be possible to find appropriate reusable components.
- Reusable components should be trustworthy (that the component will behave as specified in the requirements)
- Components should have associated documents to help the code specialist understand them and adapt them to new application.

Summary

In this module, you learned:

- Effective Principles
- Reuse Mindset
- Reuse Requirements

End of Lesson Questions

Design with reuse involves designing software around good _____ and existing components

a. bootleg
b. design
c. framework
d. architecture

25


MODULE 5


Implementing Software Reuse
Software Reuse in the Naval Open Architecture (CLE XXX)

26

Lesson Objectives

- The focus of this module is to introduce a process for implementing Software Reuse.
- Students who successfully complete this module will be able to:
  - Understand the organizational and technological enablers
  - Describe strategy to implement software reuse
  - Understand the importance of reuse metrics
  - Know how to use the SHARE repository

Enablers for Software Reuse

- The first step in setting up a reuse program is to identify and develop the enablers for Software Reuse, which consists of:
  - Organizational
    - culture, structure, polices
  - Technological
    - COTS, Component-based engineering, domain engineering

27

Example of Organizational Enablers

- Culture - assumptions, values, norms and tangible signs (artifacts) of organization's members and their behaviors
- People - Must be properly motivated and given a good understand of reuse
- Structure - functional, project, matrix, combo
- Reuse Domain - where is software reuse being used
- Reuse Potential - opportunity + ability
- Reuse Capability - ability to harness potential
- Polices and Procedures - must foster an environment for reuse

Example of Technological Enablers

- Software reuse is implemented in a number of different ways
  - Component-based Software Engineering (CBSE)
  - Service-oriented systems
  - Enterprise Application Framework (e.g. Enterprise Resource Planning systems)
  - Non developmental Items (NDI)

28

Examples of Enabling Technology

- Reuse libraries
- Generators
- Language-based systems
- Computer Aided Software Engineering tools
- Application templates
- Parameterized systems
- Software architectures
- Software schemas
- Transformational systems
- Virtual Machines Enterprise Application Framework

Knowledge Review

- Name two enablers for software implementation?

Strategy for adoption of reuse

- The second step is to establish a strategy to gain acceptance and institutionalization of Software Reuse.
- The goal is to maximize the benefits of software reuse, taking into consideration
  - Resources
  - Personnel
  - Activities
  - Desired outcomes

Example of a strategy

1. Initiate reuse program development
2. Define reuse program
3. Establish reuse adoption goals
4. Analyze reuse adoption strategies
5. Develop reuse action plan
6. Implement and monitor reuse program

30

Summary of Key Implementation Guidelines

- Ensure Organizational and Technological factors are supported
- Ensure you are designing for Reuse
- Document the software components
- Get certified (Collect metrics during project)

Why measure reuse?

- Benchmarking
  - Use for comparison purposes
- Influencing behavior
  - Provide incentives to adopt reuse
- Decision-making
  - Need data upon which to make informed decisions regarding a reuse program

Types of reuse metrics

Adapted from W. C. Lim, *Managing Software Reuse*, Upper Saddle River, N.J.: Prentice Hall, 1998, p. 301.

Important caveats

- There is no one-size-fits-all set of reuse metrics
  - The set of metrics should address the goals and needs of the organization or subunit
  - The set of metrics should address, for each reuse role, the data needs for decision making
- It is possible to misuse reuse metrics
  - Can over or understate benefits and costs of reuse

32

Berns' Six-Stage Reuse Measurement Process

1. Define and clarify goals
2. Identify reuse questions (that relate to the goals)
3. Identify reuse metrics (to answer the questions)
   - Think in terms of who, when, where, what, and how
4. Gather reuse metrics
5. Examine reuse metrics
6. Act on basis of reuse metrics

A Resource for NOA

- Software, Hardware Asset Reuse Enterprise (SHARE) repository
  - part of the Navy's Open Architecture (OA) approach to developing open, components that include reusable software applications as a core principle

Software, Hardware Asset Reuse Enterprise (SHARE) repository

- Provides a capability for discovering, accessing, sharing, managing, and sustaining reusable assets
- Consists of an asset library and a card catalog
- Currently contains only one type of software specific to surface ships
- No immediate access to assets in the repository

Summary

In this lesson, you learned:

- describe ways to implement software reuse
- describe and apply examples of software reuse applications in NOA
- Introduce the SHARE repository

End of Module Questions

- What are some organizational enablers
- What are some technological enables
- What is does SHARE stand for

CONGRATULATIONS

You have successfully completed Software Reuse in the Naval Open Architecture Continuous Learning Module (CLM)

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Estublier, J. and Vega, G., “Reuse and variability in large scale applications,” in Proceedings of the 10th European Software Engineering Conference, ACM, Lisbon, Portugal, Sept. 2005, pp. 316-325.

Griss, M. L. and Wosser, A., “Making Reuse Work at Hewlett-Packard,” IEEE Software, Jan. 1995, pp. 105-107.

Guertin, N., Presentation to the DoD Open Technology Conference, Arlington, VA, March 14, 2007.

McIlroy, M.D., “Mass produced software components,” in Naur, P. and Randell, B., eds., Report on the NATO Conference on Software Engineering, NATO Scientific Affairs Division, Brussels, Belgium, 1968, pp. 138-150.

Pepe, M., “Governments reduce, reuse, recycle code.” CRN 23 Jul 2001: 57-58. URL: http://www.crn.com/government/18815117 [last accessed April 2008].

Tomer, A., Goldin, L., Kuflik, T., Kimchi, E., and Schach, S.R., “Evaluating Software Reuse Alternatives: A Model and its Application to an Industrial Case Study,” IEEE Transactions on Software Engineering, 30, 9, Sept. 2004, pp. 601-612.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. CAPT Red Hoover
   SSC Charleston
   Charleston, SC

4. CDR Scott Heller
   SSC Charleston
   Charleston, SC

5. Dr. Bret Michael
   Naval Postgraduate School
   Monterey, CA

6. Dr. Man-Tak Shing
   Naval Postgraduate School
   Monterey, CA